

Работа №3. Представления, хр. процедуры, триггера и курсоры

Представления

1. Создать представление, отображающее все книги и читателей, о которых найдены записи в журнале с заданной даты по заданную дату.

2.

```
CREATE VIEW BooksAndReaders AS
SELECT          CLIENTS.ID    AS    CLINT_ID,  CLIENTS.FIRST_NAME,
CLIENTS.LAST_NAME,
               CLIENTS.PATHER_NAME, BOOKS.NAME
FROM            JOURNAL INNER JOIN CLIENTS ON JOURNAL.CLIENT_ID=
CLIENTS.ID
               INNER JOIN  BOOKS ON  JOURNAL.BOOK_ID= BOOKS.ID
               where  (      JOURNAL.DATE_BEG>=TO_DATE  ('1-MAY-2020')  AND
JOURNAL.DATE_BEG<=TO_DATE ('11-MAY-2020'))
               OR    (      JOURNAL.DATE_END>=TO_DATE  ('1-MAY-2020')  AND
JOURNAL.DATE_END<=TO_DATE ('11-MAY-2020'))
               OR    (      JOURNAL.DATE_RET>=TO_DATE  ('1-MAY-2020')  AND
JOURNAL.DATE_RET<=TO_DATE ('11-MAY-2020'));
```

	CLINT_ID	FIRST_NAME	LAST_NAME	PATHER_NAME	NAME
1	41	fil0	sam	1Path	R Riordan PJ
2	41	fil0	sam	1Path	R Riordan PJ
3	41	fil0	sam	1Path	Holy book
4	41	fil0	sam	1Path	Holy book
5	41	fil0	sam	1Path	ToDelete2
6	41	fil0	sam	1Path	ToDelete2
7	41	fil0	sam	1Path	1BookRussian

3. Создать представление, отображающее всех читателей и количество книг, находящихся у них на руках.

```
CREATE VIEW CountReadersBooks AS
SELECT  CLIENTS.ID, CLIENTS.FIRST_NAME, CLIENTS.LAST_NAME,
COUNT( JOURNAL.CLIENT_ID) AS BOOKS_CNT
FROM      CLIENTS FULL JOIN  JOURNAL  ON  JOURNAL.CLIENT_ID=
CLIENTS.ID
GROUP BY  CLIENTS.ID,  CLIENTS.FIRST_NAME,  CLIENTS.LAST_NAME,
JOURNAL.CLIENT_ID,JOURNAL.DATE_RET
HAVING JOURNAL.DATE_RET IS NULL;
```

	ID	FIRST_NAME	LAST_NAME	BOOKS_CNT
1	41	fil0	sam	6
2	3	dima	varl	0
3	1	dasha	petrovna	3
4	21	lchanged	lchanged	0
5	81	newww	newww	0
6	2	marina	alekseeva	4

Хранимые процедуры

- без параметров:
 1. Создать хранимую процедуру, выводящую все книги и среднее время, на которое их брали, в днях.

```

create or replace PROCEDURE AverageBookDuration Is
BEGIN
for t in (
SELECT BOOKS.ID,BOOKS.NAME,
COUNT(JOURNAL.BOOK_ID) AS BOOKS_AMOUNT,
SUM(TRUNC(JOURNAL.DATE_RET)- TRUNC( JOURNAL.DATE_BEG)) AS
DAYS_D
FROM BOOKS FULL JOIN JOURNAL ON JOURNAL.BOOK_ID=
BOOKS.ID
GROUP BY BOOKS.ID, BOOKS.NAME, JOURNAL.BOOK_ID,
JOURNAL.DATE_RET,JOURNAL.DATE_BEG
HAVING JOURNAL.DATE_RET IS NOT NULL)
loop
IF (t.BOOKS_AMOUNT!=0)THEN
DBMS_OUTPUT.PUT_LINE('BOOK_ID: '||t.ID||' | BOOK_NAME: '||t.NAME|| '
| AVG_DAYS: '||TRUNC(t.DAYS_D/t.BOOKS_AMOUNT));
ELSE
DBMS_OUTPUT.PUT_LINE('BOOK_ID: '||t.ID||' | BOOK_NAME: '||t.NAME|| '
| AVG_DAYS: '||0);
END IF;
end loop;
END AverageBookDuration;

```

```

Running dbCorrection%22test_2ygr - Log
C:\
Connecting to the database first_try.
BOOK_ID: 4 | BOOK_NAME: High N Dry Math | AVG_DAYS: 1
BOOK_ID: 2 | BOOK_NAME: Holy book | AVG_DAYS: 10
BOOK_ID: 1 | BOOK_NAME: 8 Second P2 | AVG_DAYS: 82
BOOK_ID: 1 | BOOK_NAME: 8 Second P2 | AVG_DAYS: 1
BOOK_ID: 10 | BOOK_NAME: toDelete2 | AVG_DAYS: 15
BOOK_ID: 3 | BOOK_NAME: 10 Rowling HP | AVG_DAYS: 3
Process exited.
Disconnecting from the database first_try.

```

- с входными параметрами:

1. Создать хранимую процедуру, имеющую два параметра «книга1» и «книга2». Она должна возвращать клиентов, которые вернули «книгу1» быстрее чем «книгу2». Если какой-либо клиент не брал одну из книг — он не рассматривается.

```

create or replace PROCEDURE Client_Who_Faster_First2 (book1 in NUMBER,book2 IN
NUMBER)
IS
BEGIN
for j in (
SELECT CLIENTS.ID, CLIENTS.FIRST_NAME, CLIENTS.LAST_NAME from
CLIENTS CROSS JOIN BOOKS where
(SELECT TRUNC(DATE_RET)-TRUNC(DATE_BEG) FROM JOURNAL WHERE
BOOK_ID=BOOK1 AND CLIENT_ID=CLIENTS.ID)<
(SELECT TRUNC(DATE_RET)-TRUNC(DATE_BEG) FROM JOURNAL WHERE
BOOK_ID=BOOK2 AND CLIENT_ID=CLIENTS.ID))
LOOP
DBMS_OUTPUT.PUT_LINE('CLIENT_ID: '||j.ID||' | FIRST_NAME: '||j.FIRST_NAME||'
| LAST_NAME: '||j.LAST_NAME);
end LOOP;
END Client_Who_Faster_First2;

```

```

C:\
Connecting to the database first_try.
CLIENT_ID: 1 | FIRST_NAME: Sasha | LAST_NAME: petersona
Process exited.
Disconnecting from the database first_try.

```

- с выходными параметрами:

1. Создать хранимую процедуру с входным параметром «книга» и двумя выходными параметрами, возвращающими самое большое время, на которое брали книгу, и читателя, поставившего рекорд.

```
create or replace PROCEDURE Longest_Taken_Book
(book1 in NUMBER, longClient out number, longTime out number)
IS
BEGIN
SELECT CLIENT_ID INTO longClient FROM JOURNAL WHERE Book_id=book1 AND
DATE_RET IS NOT NULL
and TRUNC(DATE_RET)-TRUNC(DATE_BEG)=
(SELECT MAX (TRUNC(DATE_RET)-TRUNC(DATE_BEG)) FROM JOURNAL WHERE
Book_id=book1 AND DATE_RET IS NOT NULL);
SELECT MAX (TRUNC(DATE_RET)-TRUNC(DATE_BEG)) INTO longTime FROM
JOURNAL WHERE Book_id=book1 AND DATE_RET IS NOT NULL;
END Longest_Taken_Book ;
```

Variable	Value
LONG...	2
LONG...	

Триггера

- Триггера на вставку:
 1. Создать триггер, который не позволяет добавить читателя с номером паспорта, который уже есть у существующего читателя.
create or replace trigger createClient
before insert on CLIENTS
for each row
DECLARE amount NUMBER;
begin
SELECT COUNT (*)into amount from clients where
PASSPORT_NUM=:new.PASSPORT_NUM AND
PASSPORT_SERIA=:new.PASSPORT_SERIA;
if (0<amount)
then raise_application_error(-20000,'Not Unique Passport Data');
end if;
end;
- Триггера на модификацию:
 1. Создать триггер, который не позволяет установить реальную дату возврата в журнале библиотекаря меньше, чем дата выдачи.
create or replace trigger updateJournalDateRet
BEFORE UPDATE ON JOURNAL
FOR EACH ROW
BEGIN
if (:new.DATE_RET<:old.DATE_BEG)
then
raise_application_error(-20001,'Wrong returning date');
END IF;
END;

- Триггера на удаление:
 1. Создать триггер, который при удалении строки журнала в случае, если книга не возвращена - откатывает транзакцию.

```
create or replace trigger deleteJournalBookNotReturned
before delete on JOURNAL
FOR EACH ROW
begin
if (:old.date_ret is null)
then
raise_application_error(-20000,'Book is not returned');
end if;
end;
```

Курсоры

- Хранимая процедура для расчета суммы штрафов библиотеки:

Необходимо реализовать хранимую процедуру, рассчитывающую сумму штрафов, полученную библиотекой за некоторый период времени. Хранимая процедура должна иметь два входных параметра, задающие интервал времени, и один выходной, в котором следует возвращать размер штрафа.

Предлагаемый алгоритм: создаем курсор, который пробегает по строкам журнала, реальная дата возврата которых попадает в заданный интервал. Для каждой строки рассчитываем размер штрафа и суммируем его в некоторой переменной, значение которой по окончании работы курсора будет выдано в качестве выходного параметра с общей суммой штрафов.

```
create or replace procedure Fees(day1 in DATE, day2 in DATE, FeeSize out NUMBER)
IS
sumFees NUMBER;--сумма штрафов
--создаем курсор, который пробегает по строкам журнала,
--реальная дата возврата которых попадает в заданный интервал.
CURSOR get_fees(day1 DATE, day2 DATE) IS
SELECT BOOK_ID FROM JOURNAL JOIN BOOKS ON
BOOKS.ID=JOURNAL.BOOK_ID
JOIN BOOK_TYPES ON BOOK_TYPES.ID=BOOKS.TYPE_ID
WHERE DATE_RET BETWEEN day1 AND day2
GROUP BY BOOK_ID;
--трибут %ROWTYPE предоставляет тип записи, представляющий строку
TYPE sFees IS RECORD (bookID JOURNAL.BOOK_ID%TYPE, typeID
BOOK_TYPES.ID);
sFees get_fees%ROWTYPE;
BEGIN
OPEN get_fees(01/02/20, 02/03/20);
loop
exit when get_fees%notfound;
fetch get_fees into FeeSize;
--Для каждой строки рассчитываем размер штрафа и суммируем его в некоторой
переменной,
sumFees:= sumFees+(TRUNC(DATE_RET)-TRUNC(DATE_end)*FINE);
end loop;
close get_fees;
DBMS_OUTPUT.put_line('Sum of Fees'|| FeeSize);
```

end Fees;

```
Connecting to the database first_try.  
Sum of Fees50  
Process exited.  
Disconnecting from the database first_try.
```

- Хранимая процедура для расчета трех самых популярных книг:

Необходимо реализовать хранимую процедуру, выбирающую три самые популярные книги за некоторый интервал времени. Хранимая процедура должна иметь два входных параметра, задающие интервал времени.

Предлагаемый алгоритм: создаём три переменные, хранящие идентификаторы самых популярных книг, и 3 переменные, соответственно, хранящие число их выдач. Создаем курсор, который пробегает по всем книгам, реальная дата выдачи которых попадает в заданный интервал. Для каждой книги рассчитываем количество ее выдач и, в случае если она была выдана большее число раз, нежели одна из сохраненных в наших переменных, то заменяем ее новой. По окончании работы курсора выбираем идентификаторы самых популярных книг.

```
create or replace PROCEDURE topBooks  
(DAY1 IN DATE, DAY2 IN DATE) IS
```

```
CURSOR get_top_books(DAY1 DATE, DAY2 DATE)  
IS
```

```
SELECT BOOK_ID, Books.Name, COUNT(*) AS popularity  
FROM JOURNAL JOIN BOOKS ON BOOKS.ID=JOURNAL.BOOK_ID  
WHERE DATE_BEG between DAY1 and DAY2  
GROUP BY BOOK_ID, Books.Name  
ORDER BY COUNT(*) DESC;
```

```
TYPE bookRec IS RECORD (ID JOURNAL.BOOK_ID%TYPE, Name  
BOOKS.NAME%TYPE);
```

```
book get_books%ROWTYPE;
```

--Атрибут %ROWCOUNT возвращает число строк считанных курсором на определенный момент времени.

```
BEGIN
```

```
open get_top_books;
```

```
LOOP
```

```
fetch get_top_books into name, popularity;
```

```
exit when(get_books%ROWCOUNT > 3);
```

```
DBMS_OUTPUT.enable;
```

```
DBMS_OUTPUT.PUT_LINE('; Name: ' || book.name || ' Count=' || book.popularity);
```

```
END LOOP;
```

```
END get_top_books;
```

```
Connecting to the database first_try.  
; Name: R Riordan PJ Count=7  
; Name: Holy book Count=6  
; Name: ToDelete2 Count=2  
Process exited.  
Disconnecting from the database first_try.
```